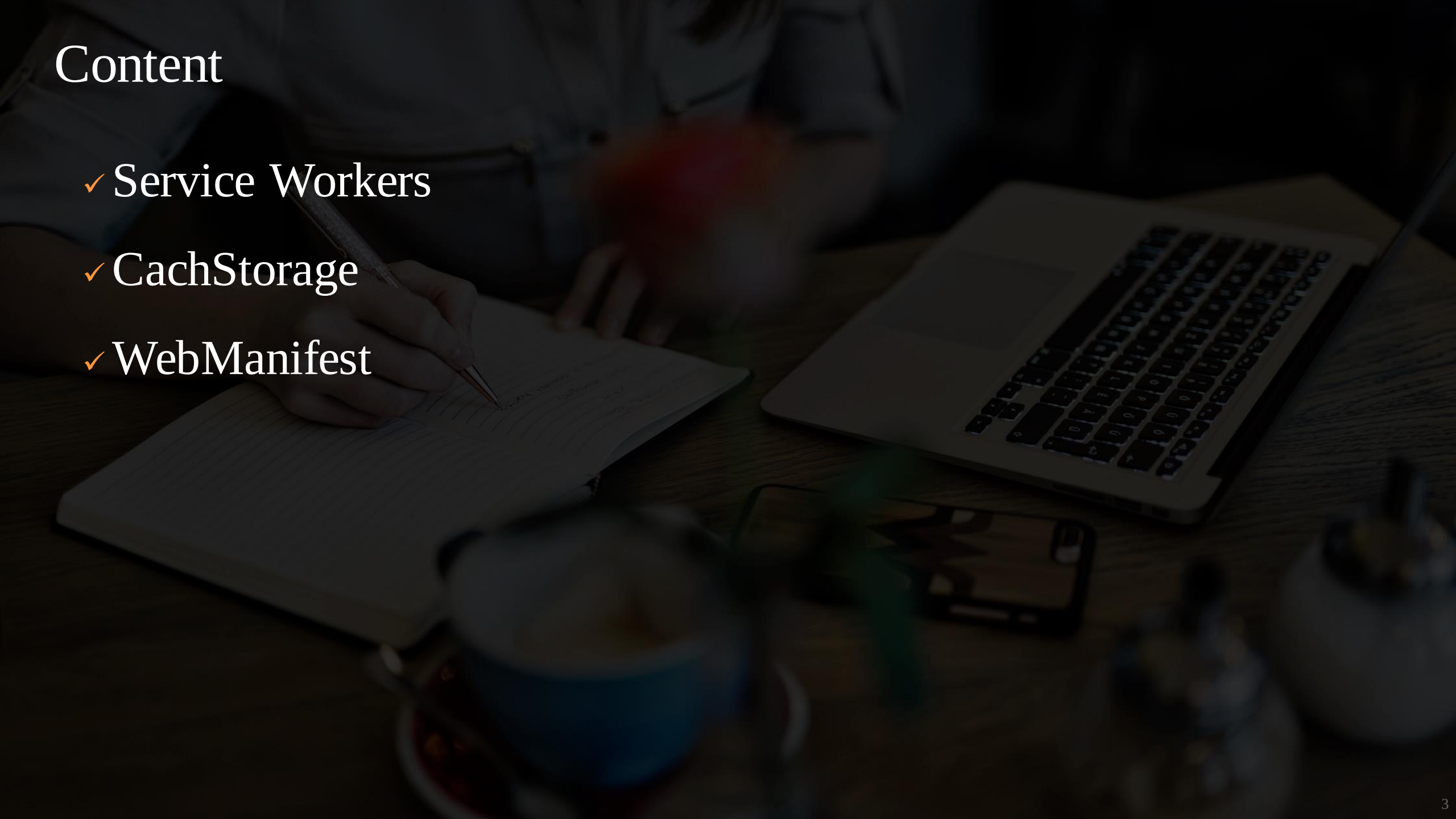
Installable

PWAs

Content

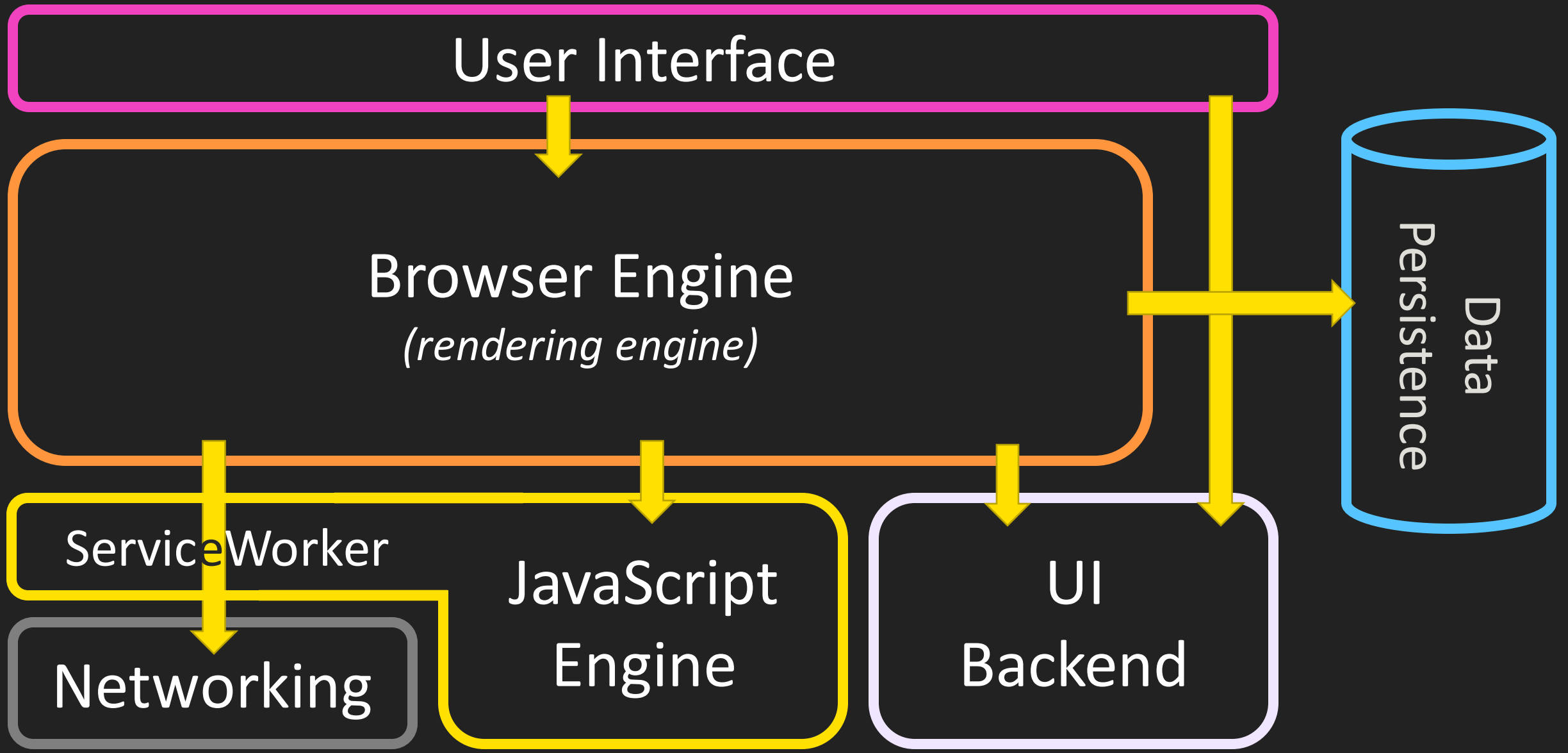
- ✓ Service Workers
- ✓ CachStorage
- ✓ WebManifest



Service Workers

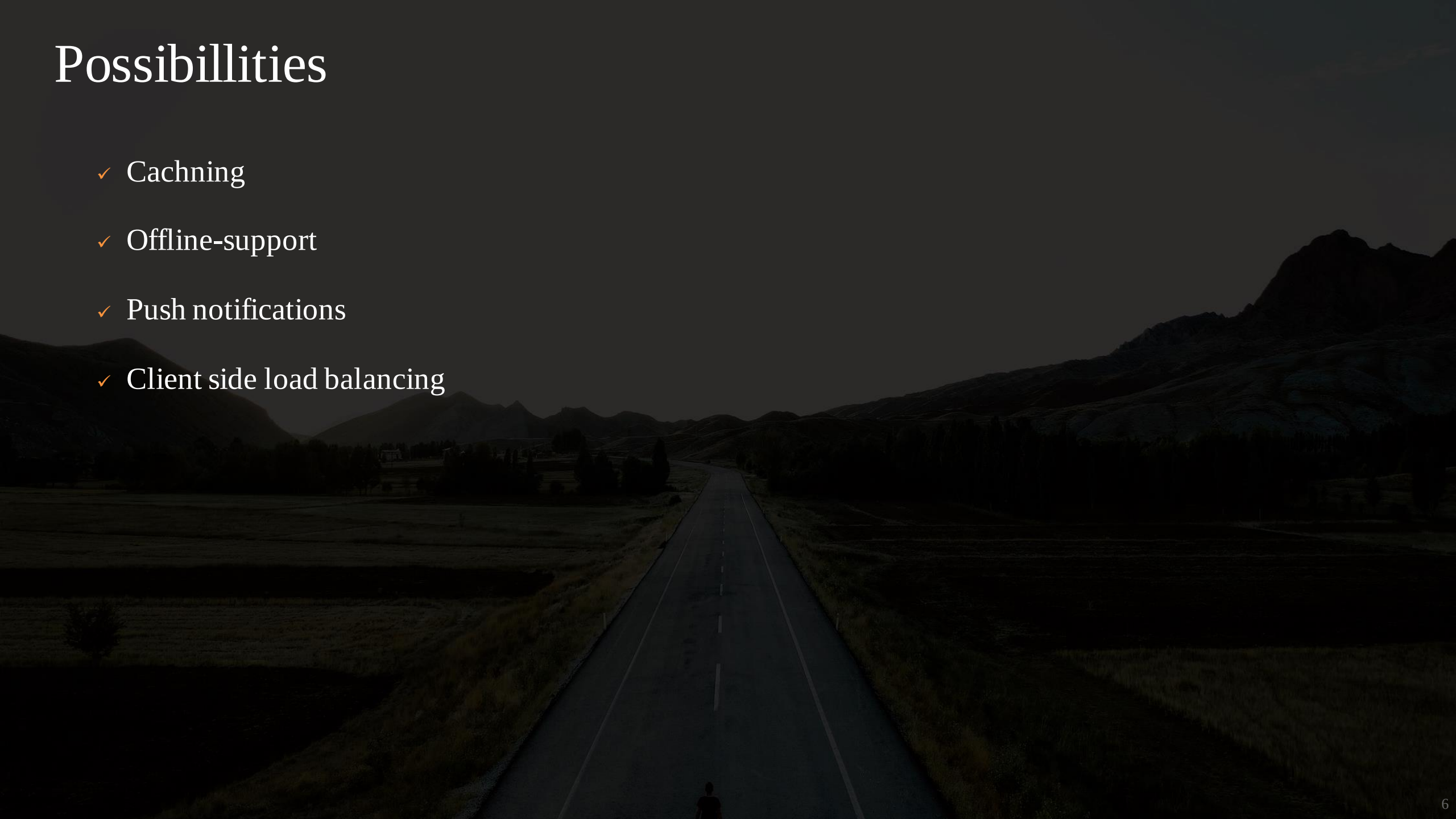
> Service worker is a programmable network proxy, allowing you to control how network requests from your page are handled.

https://developers.google.com/web/fundamentals/primers/service-workers/


User Interface
Browser Engine
(rendering engine)
Data Persistence
ServiceWorker
Networking
JavaScript Engine
UI Backend

Possibillities

- ✓ Cachning
- ✓ Offline-support
- ✓ Push notifications
- ✓ Client side load balancing



Requirements

- ✓ HTTPS (not on localhost)
- ✓ Only asynchronus calls are permitted
- ✓ Profisency in promises
- ✓ Modern Browser

- ✓ Uses a separate global scope (ServiceWorkerGlobalScope:WorkerGlobalScope)

Scopes

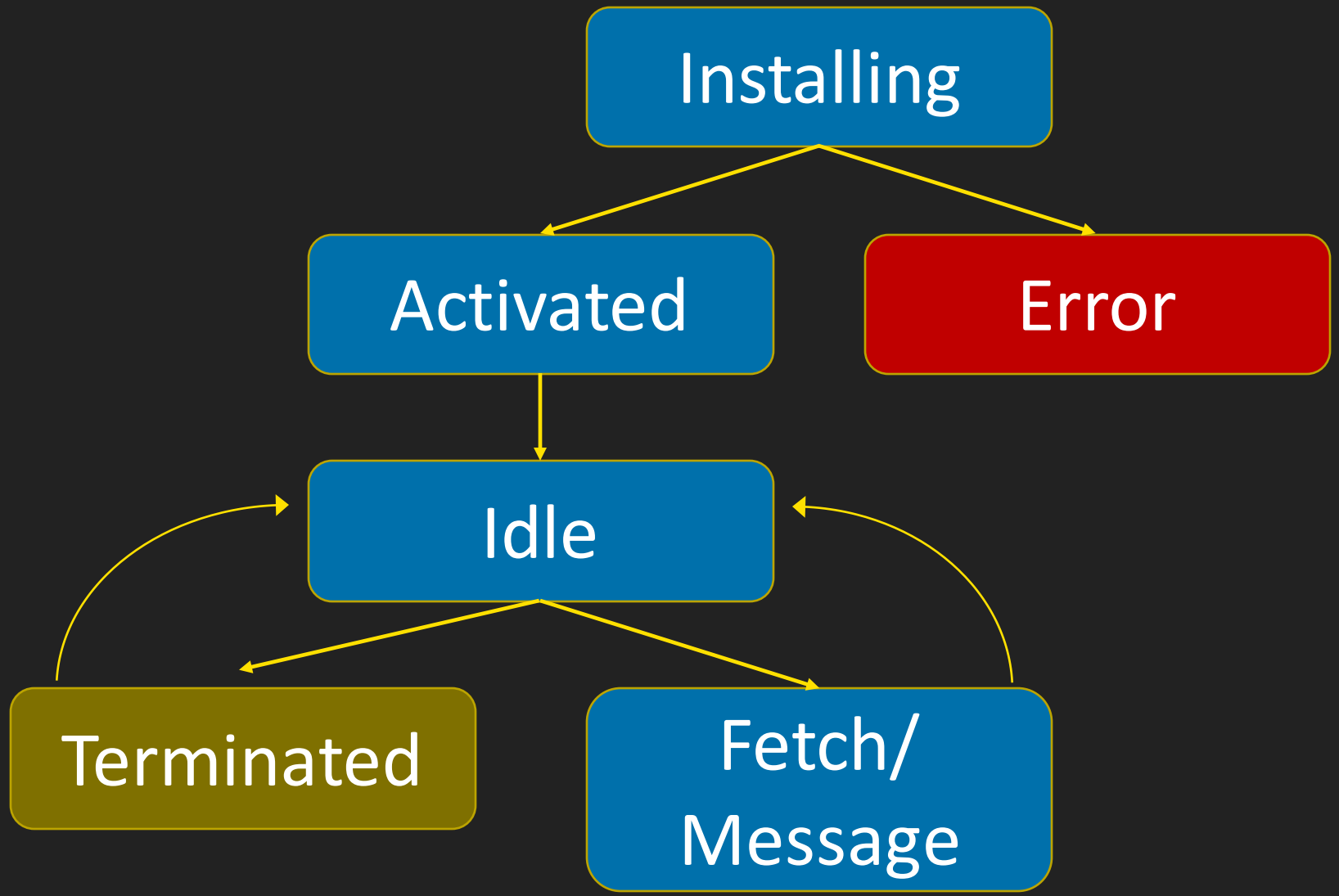
- ✓ window
- ✓ WorkerGlobalScope
- ✓ ServiceWorkerGlobalScope
- ✓ self

ServiceWorkers

- ✓ CacheStorage-API
- ✓ Fetch-API
- ✓ IndexDB
- ✓ WebSockets
- ✓ navigator/location
- ✓ setTimeout/setInterval
- ✓ (JSON)

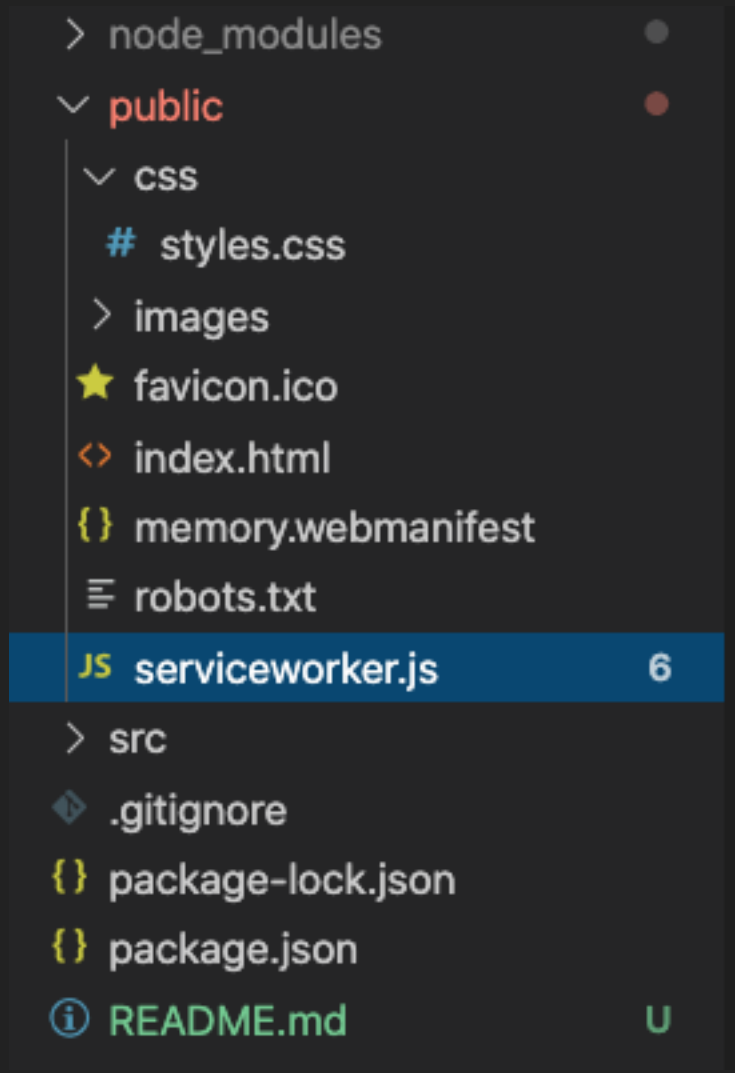
- ✓ DOM
- ✓ WebStorage

A ServiceWorkers lifecycle



https://developers.google.com/web/fundamentals/primers/service-workers/

Step 1 – Register the Service Worker



The service worker will be registered in a scope. It will only be available in that scope.

To get scope for ./, you need to place the serviceworker-file in that folder.

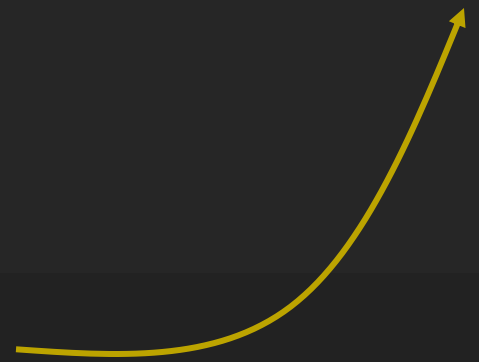
Step 1 – Register the Service Worker

"Feature detection"
Does this browser support service worker?

Make sure site is loaded before adding the service worker. Should not be necessary when using ESM

src/index.js

```
if ('serviceWorker' in navigator) {
  window.addEventListener('load', async () => {
    try {
      const registration = await navigator.serviceWorker.register('./serviceworker.js')

      console.log('ServiceWorker: Registration successful with scope: ', registration.scope)
    } catch (error) {
      console.log('ServiceWorker: Registration failed: ', error)
    }
  })
}
```



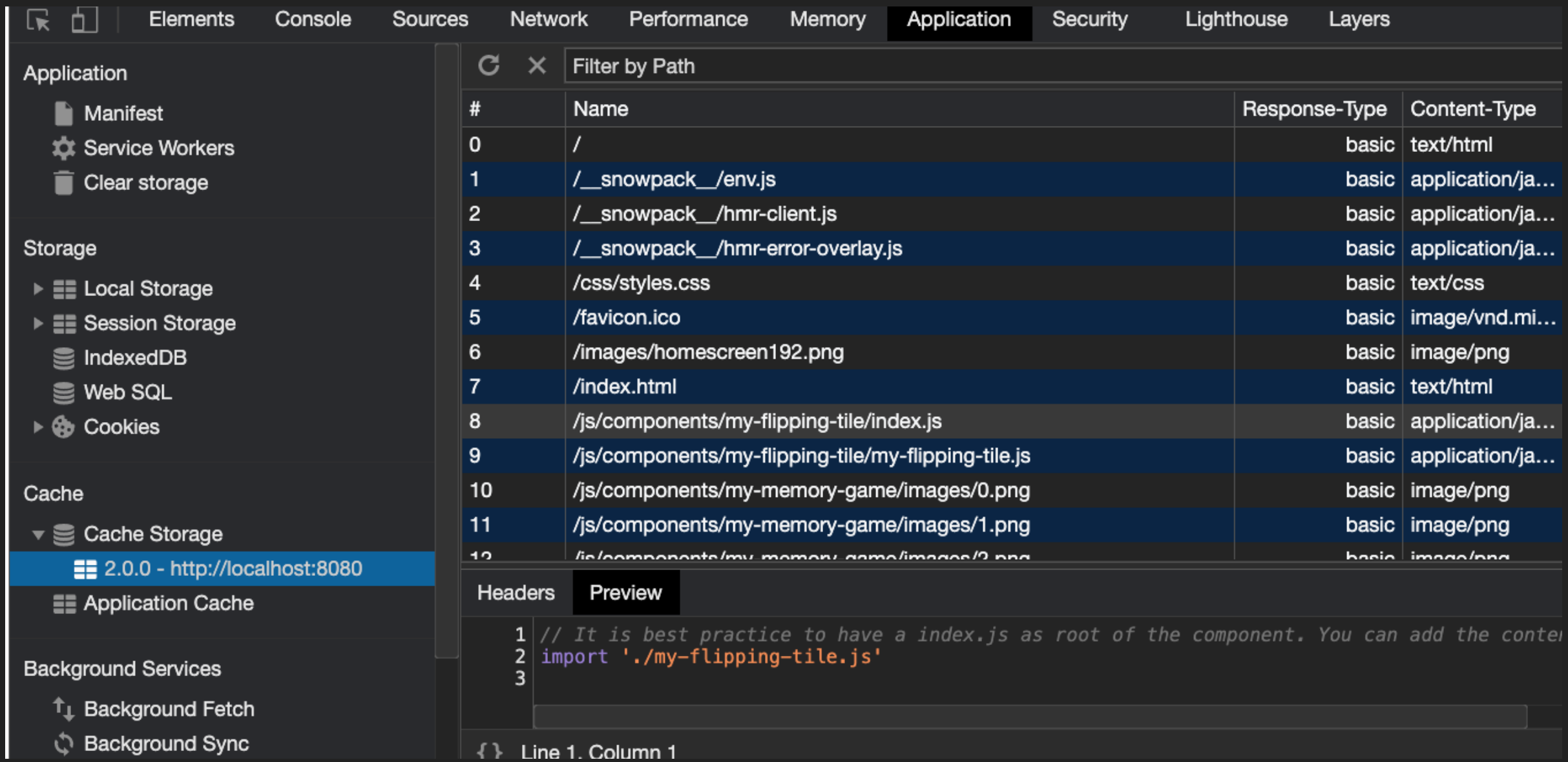
http://localhost:8080/
https://mysite.se/

serviceworker.js

```
const version = '1.0.0'

self.addEventListener('install', event => {
  console.log('ServiceWorker: Installed version ', version)
  // TODO: Cache resources needed to start
})

self.addEventListener('activate', event => {
  console.log('ServiceWorker: Activated version', version)
  // TODO: Clean up older versions of the cache
})

self.addEventListener('fetch', event => {
  console.log('ServiceWorker: Fetching')
  // TODO: Cache new resources when online and serve cached content if offline
})

self.addEventListener('message', event => {
  console.log('ServiceWorker: Got a message')
  // TODO: Handle events from the main application
})

self.addEventListener('push', event => {
  console.log('ServiceWorker: Got a push message from the server')
  // TODO: Show a notification for the user
})
```

CacheStorage

Stores Request (key) and Response (value) objects



Caching on install example

```
self.addEventListener('install', event => {
  console.info('ServiceWorker: Installed version ', version)

  /**
  * Cache assets when installing the service worker.
  *
  * @returns {Promise} Promise that resolves to undefined.
  */
  const cacheAssets = async () => {
    const cache = await self.caches.open(version)

    console.log('ServiceWorker: Caching Files')

    return cache.addAll([
      'index.html’,
      'css/styles.css’
    ])
  }

  // Make sure to wait in "installing phase" until assets are cached.
  event.waitUntil(cacheAssets())
})
```

Caching on fetch

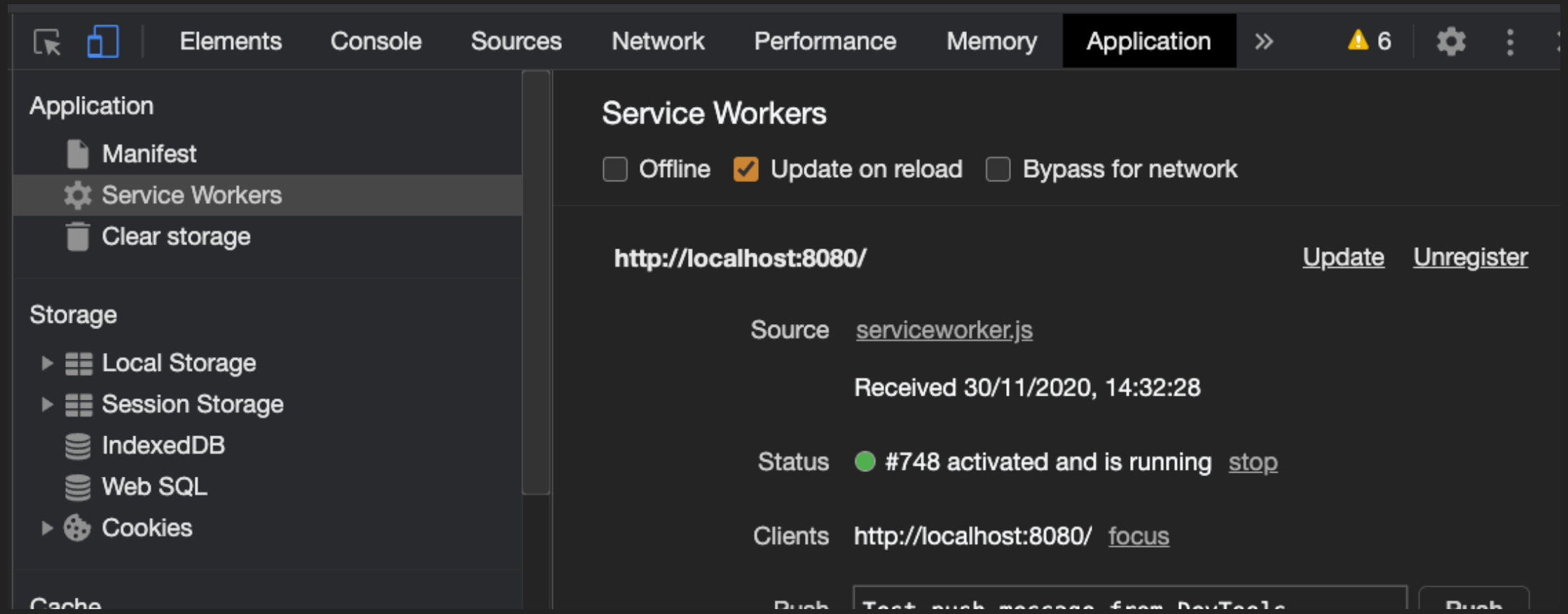
```
const cachedFetch = async request => {
  try {
    // Try to fetch the asset from the server and if success, clone the result.
    const response = await fetch(request)

    // Save the result in the cache
    const cache = await self.caches.open(version)
    cache.put(request, response.clone())

    return response
  } catch (error) {
    console.info('ServiceWorker: Serving cached result')
    return self.caches.match(request)
  }
}

self.addEventListener('fetch', event => {
  console.info('ServiceWorker: Fetching')

  event.respondWith(cachedFetch(event.request))
})
```

Need to clone since we do not wait for the cache to save the response, before we return.

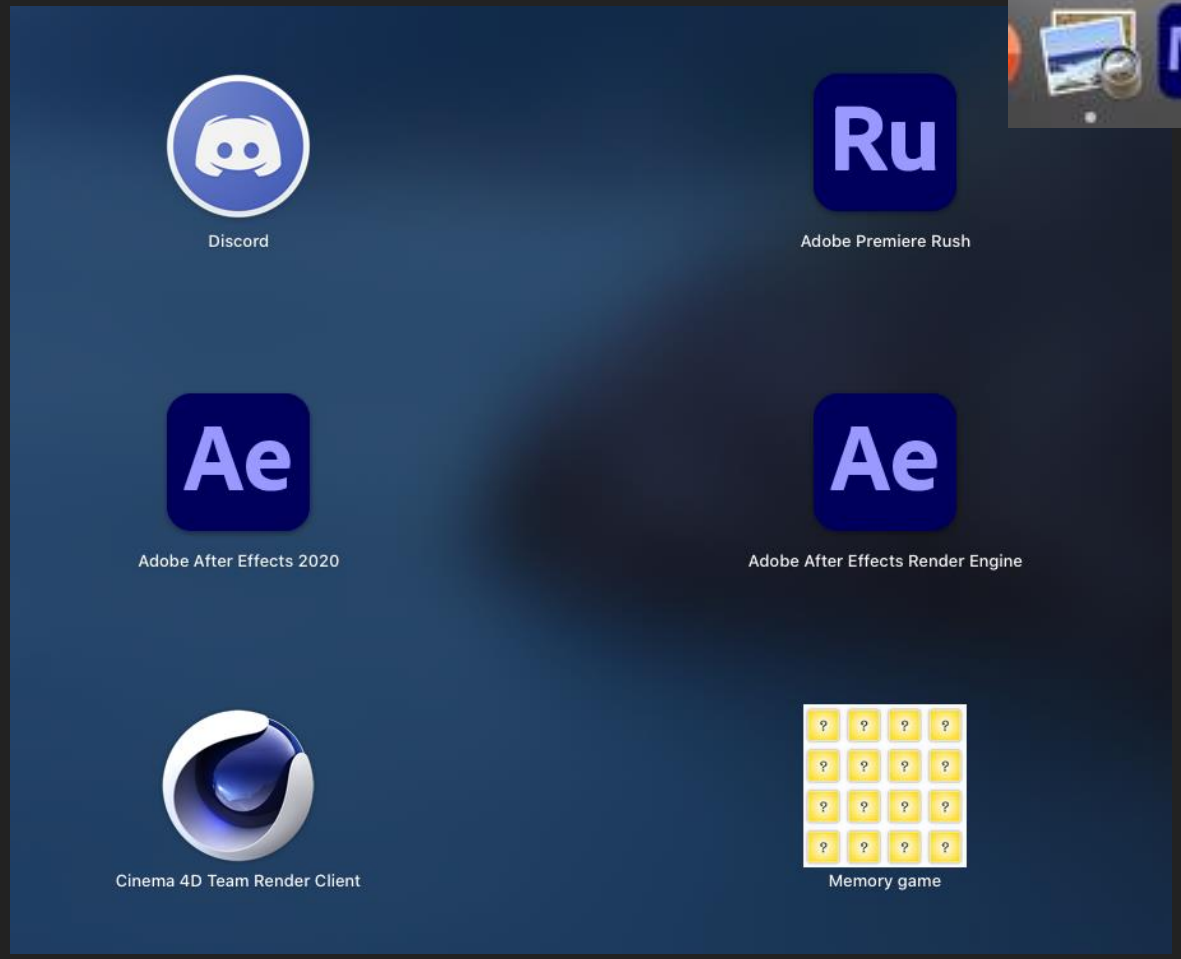
Remove Old Cached Assets

```
const removeCachedAssets = async () => {
  const cacheKeys = await self.caches.keys()

  return Promise.all(
    cacheKeys.map(cache => {
      if (cache !== version) {
        console.info('ServiceWorker: Clearing Cache', cache)
        return self.caches.delete(cache)
      }

    return undefined
  })
 )
}

self.addEventListener('activate', event => {
  console.log('ServiceWorker: Activated version', version)

  event.waitUntil(removeCachedAssets())
})
```

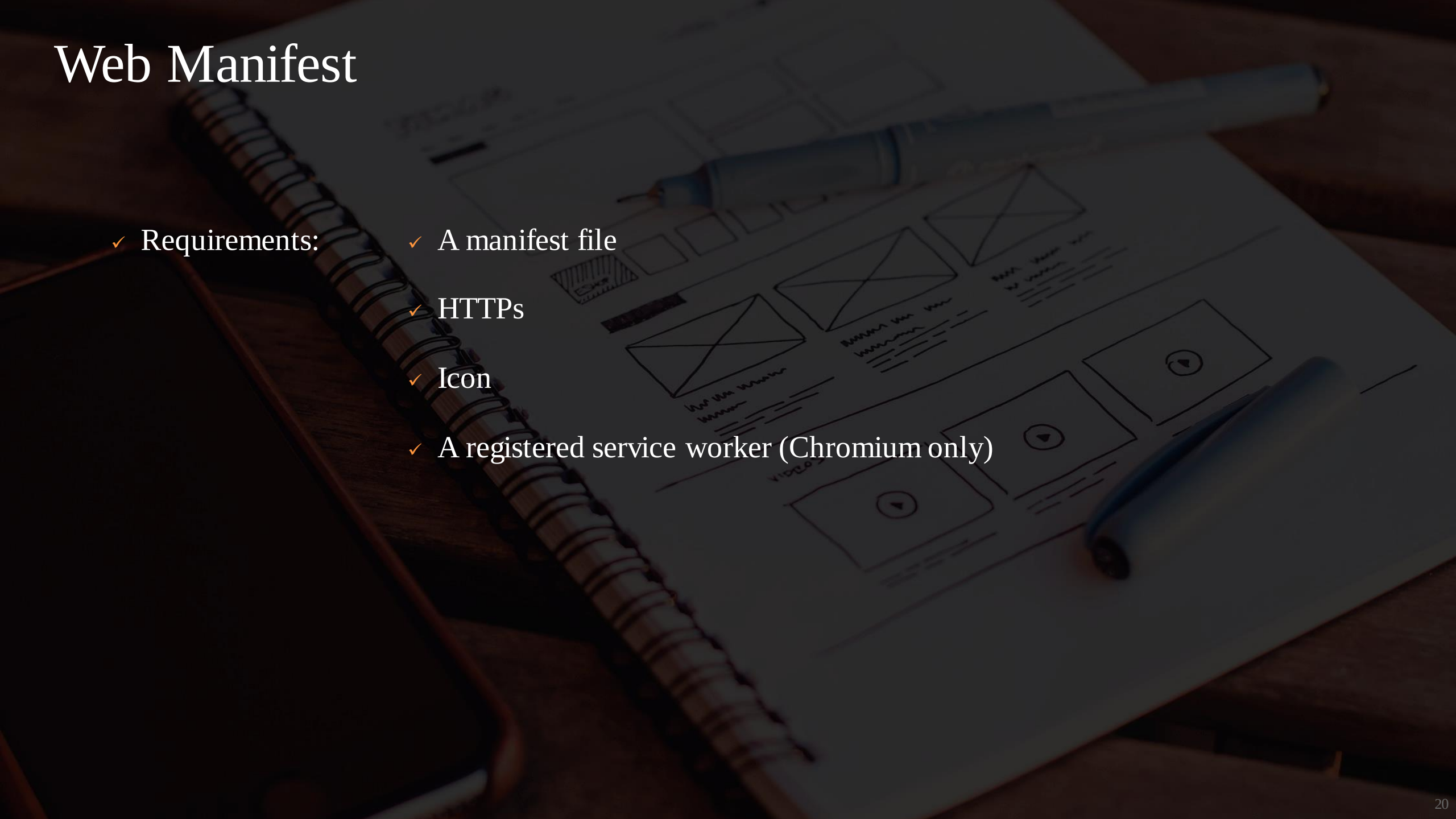
Application/ServiceWorkers



Install your PWA as an application



Web Manifest



- Requirements:
  - A manifest file
  - HTTPs
  - Icon
  - A registered service worker (Chromium only)

.webmanifest

App name

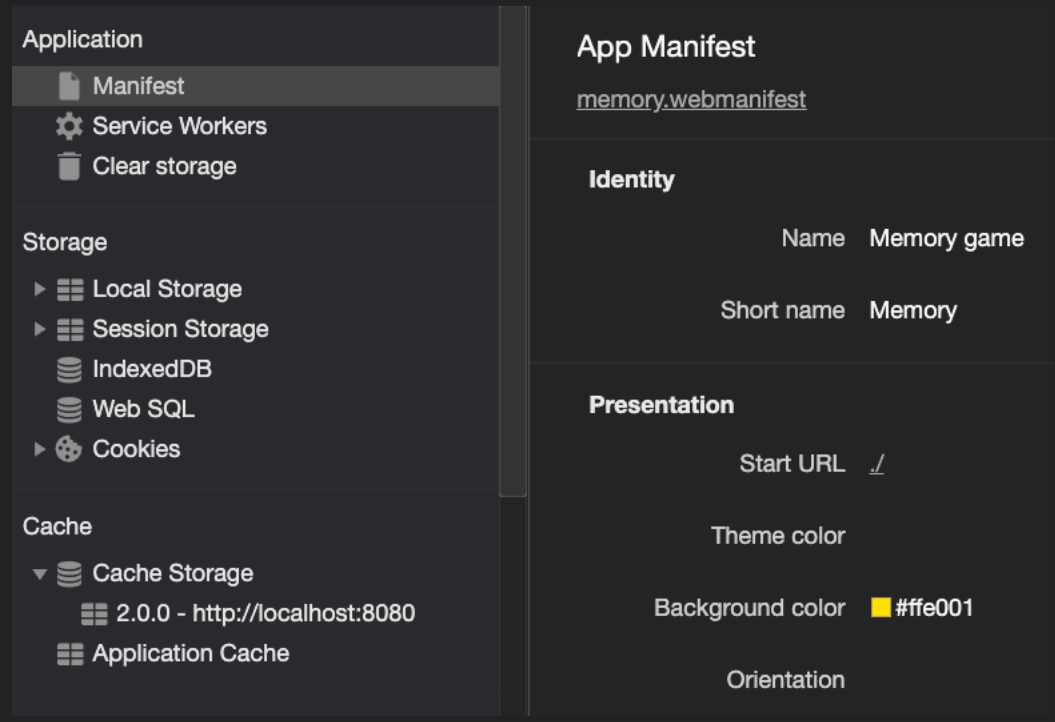
Application type:
- fullscreen
- standalone
- minimal-ui
- browser

```
{
 "name": "Memory game",
 "short_name": "Memory",
 "start_url": "./",
 "display": "standalone",
 "background_color": "#ffe001",
 "description": "A simple memory game!",
 "icons": [{
  "src": "images/homescreen48.png",
  "sizes": "48x48",
  "type": "image/png"
 }, {
  "src": "images/homescreen192.png",
  "sizes": "192x192",
  "type": "image/png"
 }]
}
```

memory.webmanifest

```
<link rel="manifest"
href="/memory.webmanifest" />
```

index.html



https://web.dev/add-manifest/

Some things to note in our environment

- ✓ Make sure to put the manifestfile and the serviceworker in the “public”-folder.
- ✓ Make sure to toggle on “Update on reload” at the service worker in the inspector
- ✓ Make sure to toggle on “Perserve log” on